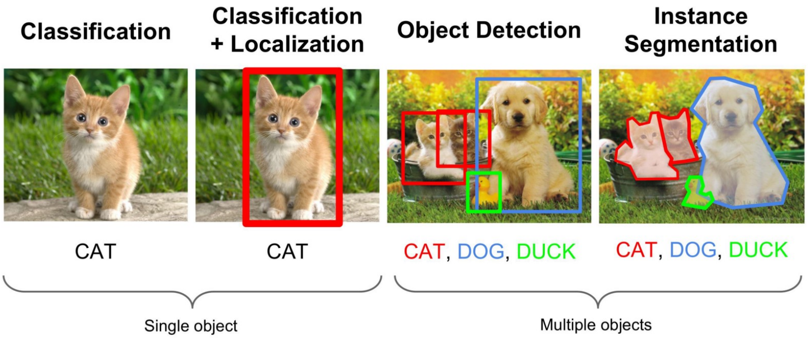
Background on Deep Learning

Nate Merrill

University of Delaware

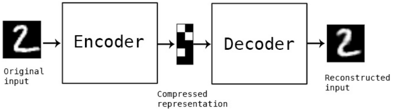
Popular Deep Learning Tasks

- Classification
- Classifcation + Bounding Box Detection
- Muti-object detection
- Pixel-wise semantic segmentation

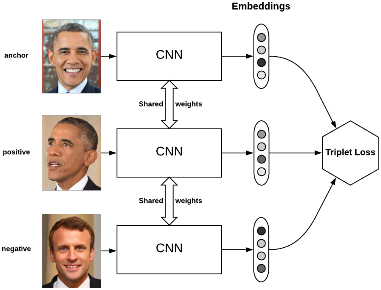


More Popular Deep Learning Tasks

- Embeddings (dimension reduction)
  - Autoencoder
  - Triplet Embedding



Autoencoder



Triplet embedding

Fully Connected Layer

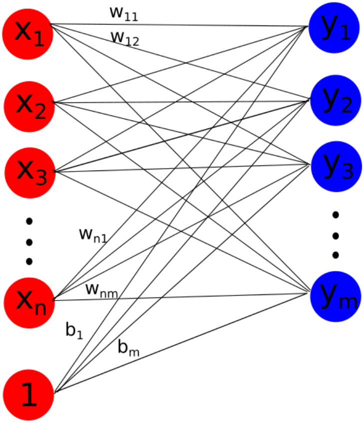
- Vector input and output
- Learned weights associated with each connection
- Can be written as a linear operation

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n & 1 \end{bmatrix}^\mathsf{T}$$

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix}^\mathsf{T}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & b_1 \\ w_{21} & w_{22} & \dots & w_{2n} & b_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} & b_m \end{bmatrix}$$

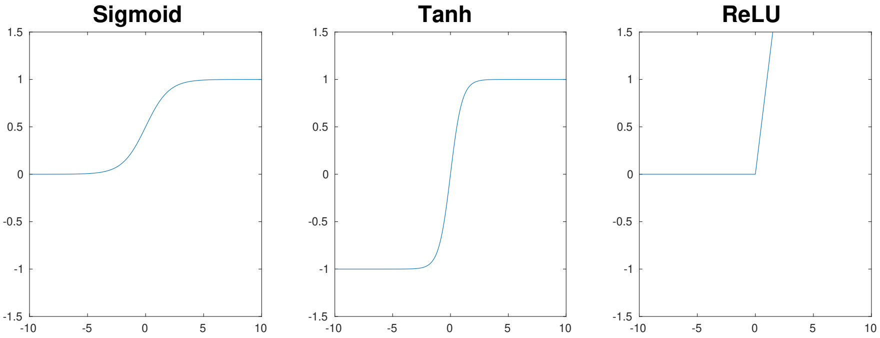
$$\mathbf{y} = \mathbf{W}\mathbf{x}$$



Nonlinear Activation

- Fully connected layer cannot model nonlinear functions
- Nonlinear activations are used to provide nonlinearity
- Key idea: they resemble a "neuron" firing

Examples:



Popular activation functions

Convolution Layer

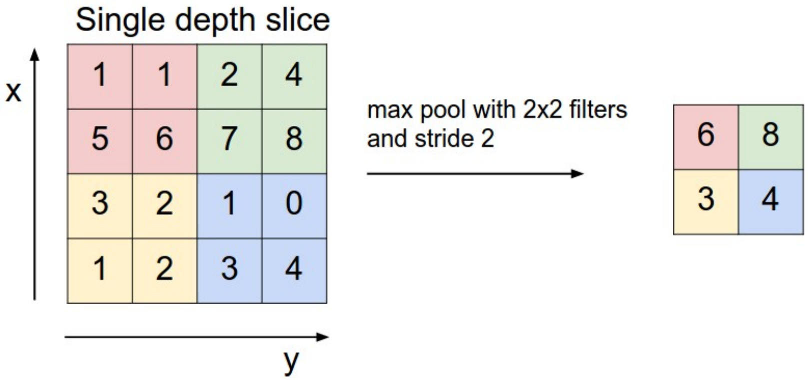
- Matrix/Tensor input and output
- Useful for image input

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} & X_{15} & X_{16} & X_{17} \\ X_{21} & X_{22} & X_{23} & X_{24} & X_{25} & X_{26} & X_{27} \\ X_{31} & X_{32} & X_{33} & X_{34} & X_{35} & X_{36} & X_{37} \\ X_{41} & X_{42} & X_{43} & X_{44} & X_{45} & X_{46} & X_{47} \\ X_{51} & X_{52} & X_{53} & X_{54} & X_{55} & X_{56} & X_{57} \\ X_{61} & X_{62} & X_{63} & X_{64} & X_{65} & X_{66} & X_{67} \\ X_{71} & X_{72} & X_{73} & X_{74} & X_{75} & X_{76} & X_{77} \end{bmatrix} * \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} & Y_{14} & Y_{15} & Y_{16} \\ Y_{21} & Y_{22} & Y_{23} & Y_{24} & Y_{25} & Y_{26} \\ Y_{31} & Y_{32} & Y_{33} & Y_{34} & Y_{35} & Y_{36} \\ Y_{41} & Y_{42} & Y_{43} & Y_{44} & Y_{45} & Y_{46} \\ Y_{51} & Y_{52} & Y_{53} & Y_{54} & Y_{55} & Y_{56} \\ Y_{61} & Y_{62} & Y_{63} & Y_{64} & Y_{65} & Y_{66} \end{bmatrix}$$

$$\begin{aligned} Y_{ij} = {} & K_{33}X_{i,j} + K_{32}X_{i,j+1} + K_{31}X_{i,j+2} + K_{23}X_{i+1,j} \\ & + K_{22}X_{i+1,j+1} + K_{21}X_{i+1,j+2} + K_{13}X_{i+2,j} \\ & + K_{12}X_{i+2,j+1} + K_{11}X_{i+2,j+2} + b \end{aligned}$$

Max Pooling Layer

- Useful for viewpoint invariance
- Similar operation to convolution



Softmax Layer

- Bounds output to [0,1]
- Sum of output is 1
- Useful for learning probability mass functions

Suppose $\mathbf{x} \in \mathbb{R}^n$, then

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Loss Function

- A scalar error metric for the network
- Necessary for training. Otherwise there is no optimization criteria

Popular examples:

Cross Entropy
Suppose $\mathbf{x} \in \{0, 1\}^n$ is the ground truth class labels and $\hat{\mathbf{x}} \in [0, 1]^n$ is the output of a *classification* network, then

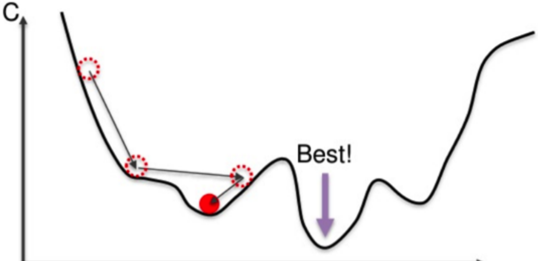
$$E(\mathbf{x}, \hat{\mathbf{x}}) = -\sum_{i=1}^{n} x_i \ln(\hat{x}_i)$$

MSE Suppose $\mathbf{x} \in \mathbb{R}^n$ is the ground truth and $\hat{\mathbf{x}} \in \mathbb{R}^n$ is the output of a *regression* network, then

$$E(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{n} ||\hat{\mathbf{x}} - \mathbf{x}||_2^2$$

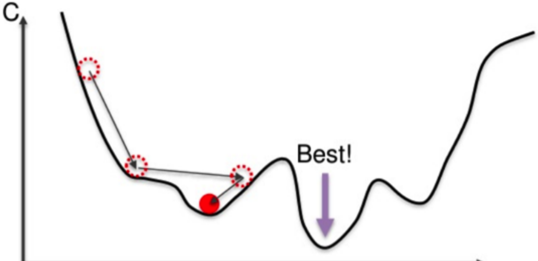
Optimization

- Suppose we have a feed-forward neural network with arbitrary tensor input $\boldsymbol{X}$, output (loss/error) E, and layers $L_1, L_2, \ldots, L_k$ (including loss layer).
- How can we optimize such a complicated function?
- There are many local minimums to get caught in



Optimization

- Suppose we have a feed-forward neural network with arbitrary tensor input $\boldsymbol{X}$, output (loss/error) E, and layers $L_1, L_2, \ldots, L_k$ (including loss layer).
- How can we optimize such a complicated function?
- There are many local minimums to get caught in
  - **Idea:** Look at each layer separately
  - Observe that $E = L_k(L_{k-1}(\ldots(L_1(\boldsymbol{X}))))$.
  - We can estimate the gradient of the entire network with the chain rule!



Optimization: Stochastic Gradient Descent

- Create copies of your network
  - Train on multiple data samples at once (i.e., a *batch*)
  - Average out the loss values across examples
  - Make sure the examples are selected *randomly*
- At each iteration, estimate the gradient of each layer w.r.t. the weights
- Update the weights with the *average* gradient over the batch with a scalar multiplier called the *learning rate* (typically $<< 1$)

Suppose we have the weights of layer L_i, at iteration j, $\boldsymbol{W}_i^{(j)}$, with layer input $\boldsymbol{X}$, learning rate η, and n examples per batch. Then the update step is:

$$\boldsymbol{W}_i^{(j+1)} = \boldsymbol{W}_i^{(j)} - \frac{\eta}{n} \sum_{k=1}^{n} \nabla L_i(\boldsymbol{W}_i^{(j)}, \boldsymbol{X}_k)$$